Building the Next Generation of Parallel Applications

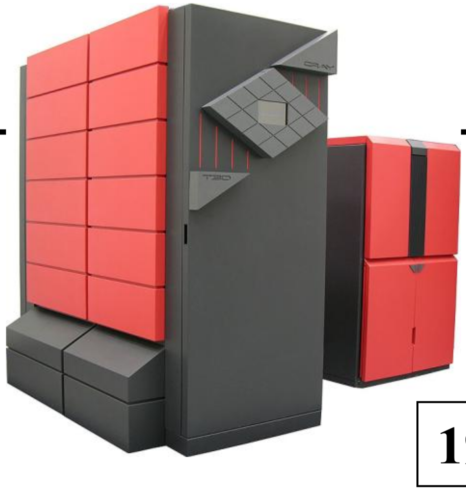
Michael A. Heroux

Scalable Algorithms Department

Sandia National Laboratories, USA

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

A Brief Personal Computing History



1988 - 1997

1993 - 2008

```
CMIC$ DO ALL VECTOR IF (N .GT. 800)
CMIC$1   SHARED(BETA, N, Y, Z)
CMIC$2   PRIVATE(I)
CDIR$ IVDEP
        do 15 i = 1, n
           z(i) = beta * y(i)
 15     continue
      endif
```

```
 #include <mpi.h>
int main(int argc, char *argv[]) {
// Initialize MPI
  MPI_Init(&argc,&argv);
  int rank, size;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
```

2008 - Present

Unification and composition:

-Vectorization

-Threading

- Multiprocessing

```
 #include <mpi.h>
#include <omp.h>
int main(int argc, char *argv[]) {
// Initialize MPI
  MPI_Init(&argc,&argv);
  int rank, size;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
…
#pragma omp parallel
{
      double localasum = 0.0;
#pragma omp for
      for (int j=0; j< MyLength_; j++) localasum += std::abs(from[j]);
#pragma omp critical
      asum += localasum;
}
```

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>

thrust::device_vector<int> vd(10, 1);
thrust::host_vector<int> vh(10,1);
```

Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have MPI_Init().
3. All future programmers will need to write parallel code.
4. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
5. DRY is not possible across CPUs and GPUs
6. GPUs are a harbinger of CPU things to come.
7. Checkpoint/Restart will be sufficient for scalable resilience.
8. Resilience will be built into algorithms.
9. MPI-only and MPI+X can coexist in the same application.
10. Kernels will be different in the future.

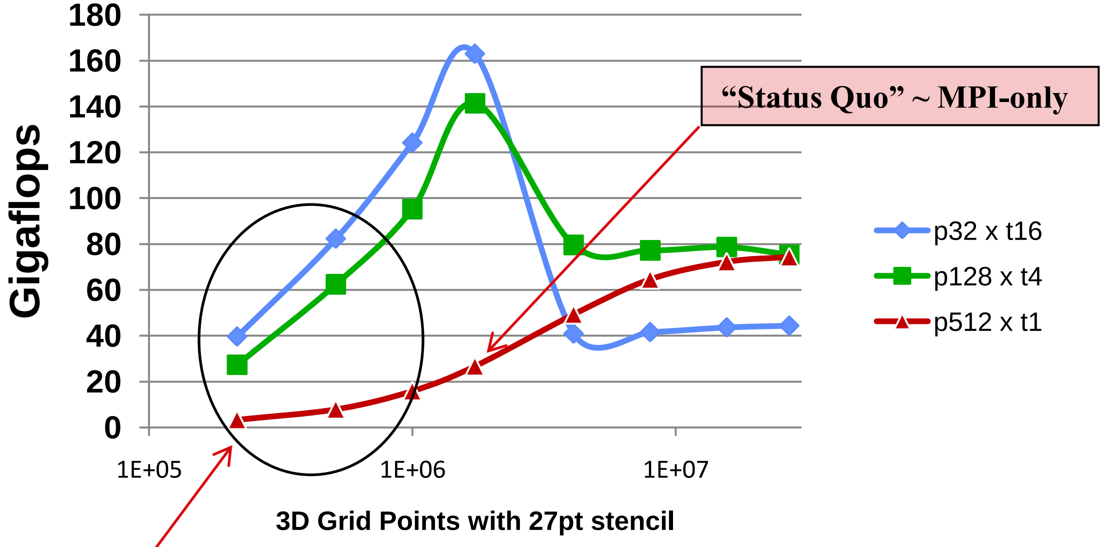
Basic Exascale Concerns: Trends, Manycore

- Stein’s Law: *If a trend cannot continue, it will stop.*

  Herbert Stein, chairman of the Council of Economic Advisers under Nixon and Ford.

- Trends at risk:
  - Power.
  - Single core performance.
  - Node count.
  - Memory size & BW.
  - Concurrency expression in existing Programming Models.

Parallel CG Performance 512 Threads
32 Nodes = 2.2GHz AMD 4sockets X 4cores



“Status Quo” ~ MPI-only

Strong Scaling Potential

Edwards: SAND2009-8196
Trilinos ThreadPool Library v1.1.

One outcome: Greatly increased interest in OpenMP

Implications

- MPI-Only is not sufficient, except … much of the time.
- Near-to-medium term:
  - MPI+[OMP|TBB|Pthreads|CUDA|OCL|MPI]
  - Long term, too?
- Long- term:
  - Something hierarchical, global in scope.
- Conjecture:
  - Data-intensive apps need non-SPDM model.
  - Will develop new programming model/env.
  - Rest of apps will adopt over time.
  - Time span: 20 years.

What Can we Do Right Now?

- Study why MPI was successful.
- Study new parallel landscape.
- Try to cultivate an approach similar to MPI.

MPI Impresssions

MPI: It Hurts So Good

- Observations
  - “assembly language” of paral
  - lowest common denomin
    - portable across arc
  - upfront effort r
    - system
    - ex
- Co

Dan Reed, Microsoft

Workshop on the Road Map for the Revitalization of High End Computing

June 16-18, 2003

So What Would Life Be Like Without MPI?

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

Serial C

```
long fib_serial(long n)
{
    if (n < 2) return n;
    return fib_serial(n-1) + fib_serial(n-2);
}
```

Cilk++

```
long fib_parallel(long n)
{
    long x, y;
    if (n < 2) return n;
    x = cilk_spawn fib_parallel(
    y = fib_parallel(n-2);
    cilk_sync;
    return (x
}
```

OpenMP 3.0

```
long fib_parallel(long n)
{
    long x, y;
    if (n < 2) return n;
    #pragma omp task default(none) shared(x,n)
    {
        x = fib_parallel(n-1);
    }
    y = fib_parallel(n-2);
    #pragma omp taskwait
    return (x+y);
}
```

X10

```
public class
    var
```

```
def fib_ser
{
    var x,y: n.
    if (n < 2) the
    cobegin {
        x=fib_serial(
        y=fib_serial(n
    }
    return x+y;
}
```

```
fib_parallel(n: ZZ): ZZ requi
    if n < 2 then n else fib_par
```

Tim Stitts, CSCS

SOS14 Talk

March 2010

Looking Forward to a New Age of Large-Scale Parallel Programming and the Demise of MPI

...hopes and dreams of an HPC educator

“ MPI is often considered the “portable assembly language” of parallel computing, …”

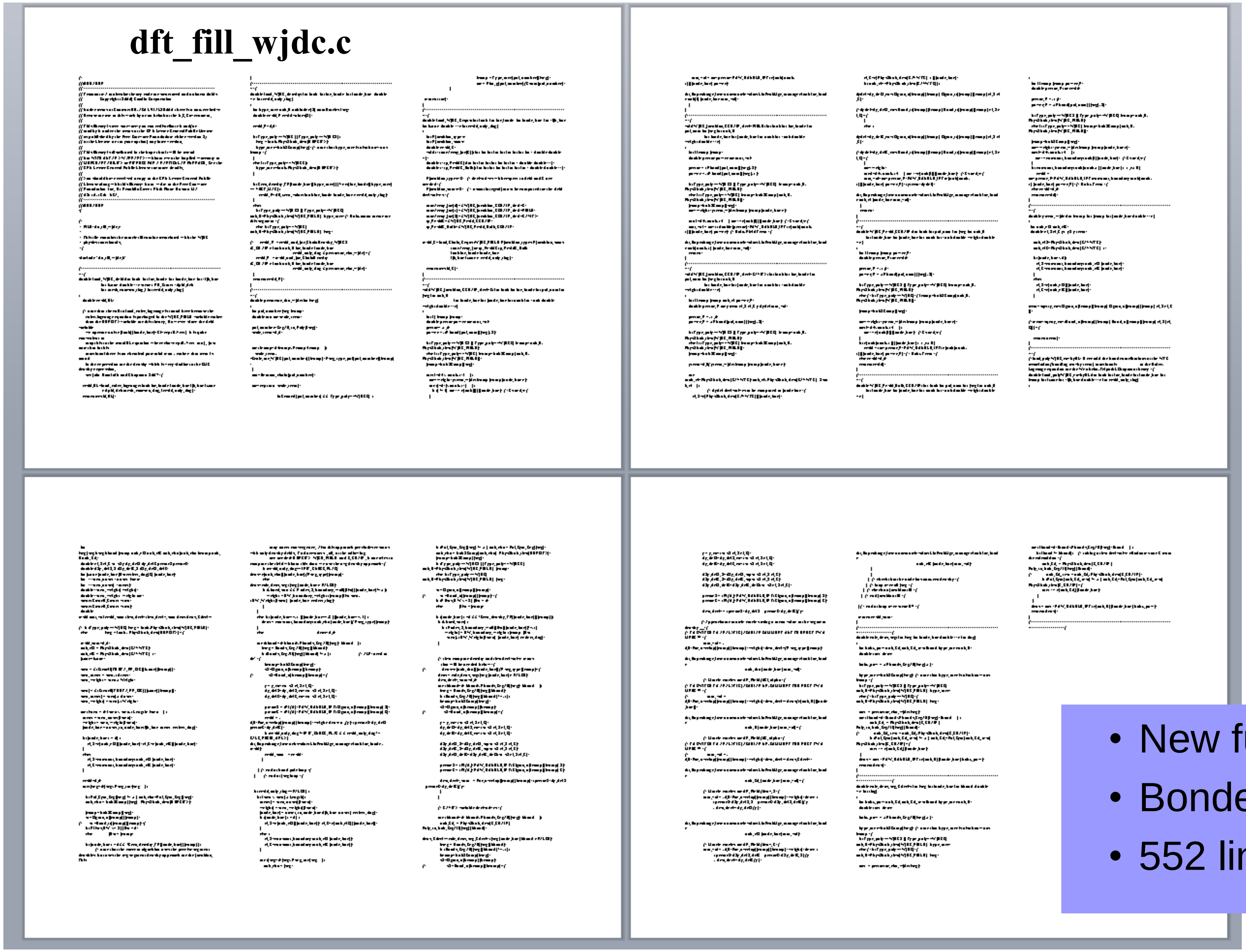
Brad Chamberlain, Cray, 2000.

3D Stencil in NAS MG

Brad Chamberlain, Cray, PPOPP’06, http://chapel.cray.com/publications/ppopp06-slides.pdf

MPI Reality

Tramonto WJDC Functional



dft_fill_wjdc.c

- New functional.
- Bonded systems.
- 552 lines C code.

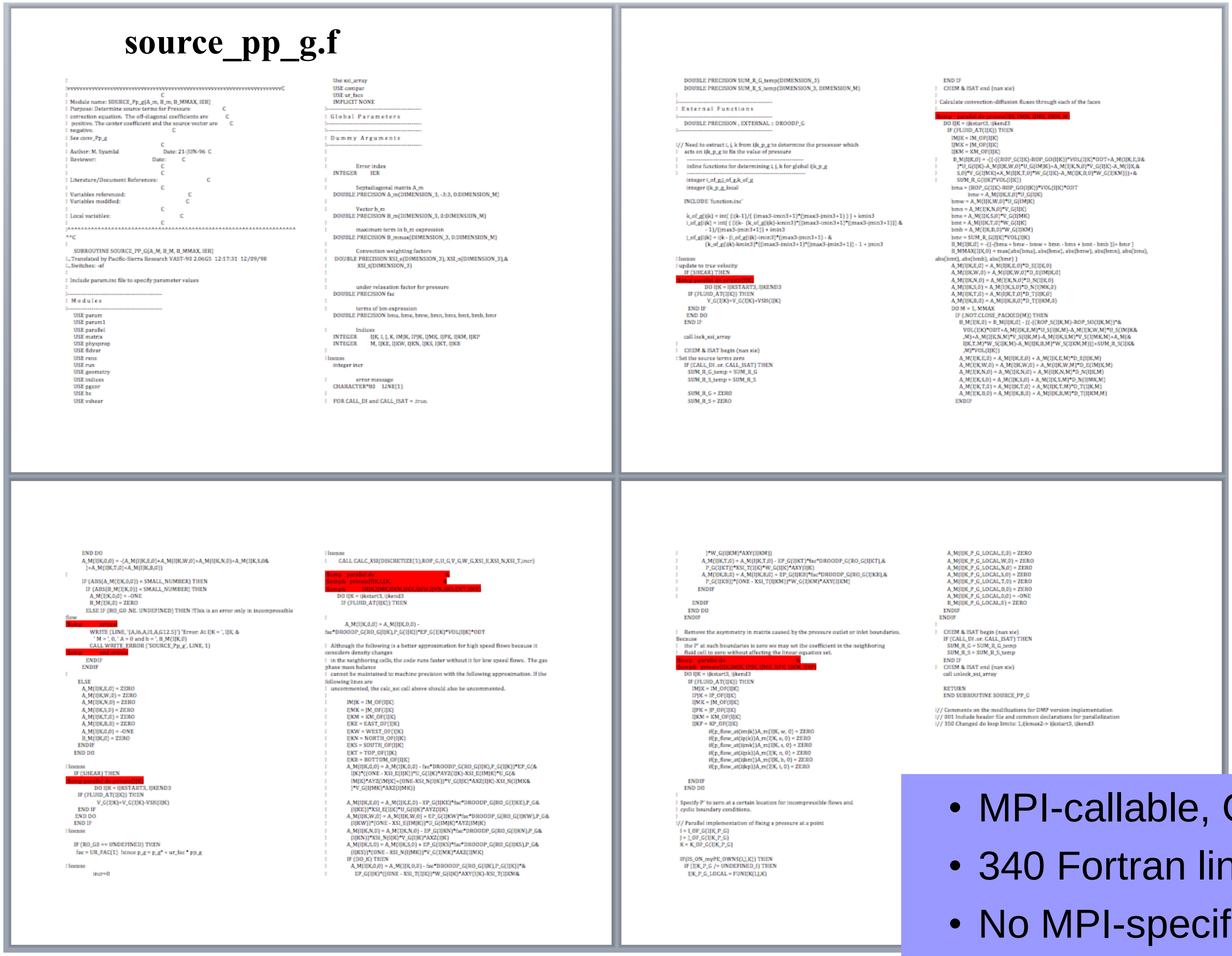
WJDC-DFT (Werthim, Jain, Dominik, and Chapman) theory for bonded systems. ***(S. Jain, A. Dominik, and W.G. Chapman. Modified interfacial statistical associating fluid theory: A perturbation density functional theory for inhomogeneous complex fluids. J. Chem. Phys., 127:244904, 2007.)*** **Models stoichiometry constraints inherent to bonded systems.**

How much MPI-specific code?

dft_fill_wjdc.c
MPI-specific
code

MFIX

Source term for pressure correction



source_pp_g.f

```
!
!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvC
!                                                          C
!  Module name: SOURCE_Pp_g(A_m, B_m, B_MMAX, IER)
!  Purpose: Determine source terms for Pressure                 C
!  correction equation.  The off-diagonal coefficients are      C
!   positive. The center coefficient and the source vector are  C
!  negative.                                          C
!  See conv_Pp_g
!                                                          C
!  Author: M. Syamlal                        Date: 21-JUN-96  C
!  Reviewer:                              Date:          C
!                                                          C
!                                                          C
!  Literature/Document References:                              C
!                                                          C
!  Variables referenced:                                   C
!  Variables modified:                                     C
!                                                          C
!  Local variables:                                  C
!                                                          C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^C
!
    SUBROUTINE SOURCE_PP_G(A_M, B_M, B_MMAX, IER)
!...Translated by Pacific-Sierra Research VAST-90 2.06G5  12:17:31  12/09/98
!...Switches: -xf
!
!  Include param.inc file to specify parameter values
!
!-----------------------------------------------
!   M o d u l e s
!-----------------------------------------------
    USE param
    USE param1
    USE parallel
    USE matrix
    USE physprop
    USE fldvar
    USE rxns
    USE run
    USE geometry
    USE indices
    USE pgcor
    USE bc
    USE vshear
    Use xsi_array
    USE compar
    USE ur_facs
    IMPLICIT NONE
!-----------------------------------------------
!   G l o b a l   P a r a m e t e r s
!-----------------------------------------------
!-----------------------------------------------
!   D u m m y   A r g u m e n t s
!-----------------------------------------------
!
!
!                 Error index
    INTEGER          IER
!
!                 Septadiagonal matrix A_m
    DOUBLE PRECISION A_m(DIMENSION_3, -3:3, 0:DIMENSION_M)
!
!                 Vector b_m
    DOUBLE PRECISION B_m(DIMENSION_3, 0:DIMENSION_M)
!
!                 maximum term in b_m expression
    DOUBLE PRECISION B_mmax(DIMENSION_3, 0:DIMENSION_M)
!
!                 Convection weighting factors
!     DOUBLE PRECISION XSI_e(DIMENSION_3), XSI_n(DIMENSION_3),&
!                 XSI_t(DIMENSION_3)
!
!
!                 under relaxation factor for pressure
    DOUBLE PRECISION fac
!
!                 terms of bm expression
    DOUBLE PRECISION bma, bme, bmw, bmn, bms, bmt, bmb, bmr

!                 Indices
    INTEGER          IJK, I, J, K, IMJK, IPJK, IJMK, IJPK, IJKM, IJKP
    INTEGER          M, IJKE, IJKW, IJKN, IJKS, IJKT, IJKB
!
!loezos
    integer incr

!                 error message
    CHARACTER*80     LINE(1)
!
!   FOR CALL_DI and CALL_ISAT = .true.
    DOUBLE PRECISION SUM_R_G_temp(DIMENSION_3)
    DOUBLE PRECISION SUM_R_S_temp(DIMENSION_3, DIMENSION_M)
!
!-----------------------------------------------
!   E x t e r n a l   F u n c t i o n s
!-----------------------------------------------
    DOUBLE PRECISION , EXTERNAL :: DROODP_G
!-----------------------------------------------

!// Need to extract i, j, k from ijk_p_g to determine the processor which
!    acts on ijk_p_g to fix the value of pressure
!     ----------------------------------------------------------
!     inline functions for determining i, j, k for global ijk_p_g
!     ----------------------------------------------------------
      integer i_of_g,j_of_g,k_of_g
      integer ijk_p_g_local

    INCLUDE 'function.inc'

      k_of_g(ijk) = int( (ijk-1)/( (imax3-imin3+1)*(jmax3-jmin3+1) ) ) + kmin3
      i_of_g(ijk) = int( ( (ijk-  (k_of_g(ijk)-kmin3)*((imax3-imin3+1)*(jmax3-jmin3+1))) &
                - 1)/(jmax3-jmin3+1)) + imin3
      j_of_g(ijk) = ijk - (i_of_g(ijk)-imin3)*(jmax3-jmin3+1) - &
                (k_of_g(ijk)-kmin3)*((imax3-imin3+1)*(jmax3-jmin3+1)) - 1 + jmin3

!loezos
!update to true velocity
    IF (SHEAR) THEN
$omp parallel do private(IJK)
              DO IJK = IJKSTART3, IJKEND3
      IF (FLUID_AT(IJK)) THEN
               V_G(IJK)=V_G(IJK)+VSH(IJK)
      END IF
     END DO
    END IF

    call lock_xsi_array
!
!   CHEM & ISAT begin (nan xie)
! Set the source terms zero
    IF (CALL_DI .or. CALL_ISAT) THEN
      SUM_R_G_temp = SUM_R_G
      SUM_R_S_temp = SUM_R_S

      SUM_R_G = ZERO
      SUM_R_S = ZERO
    END IF
!   CHEM & ISAT end (nan xie)
!
!  Calculate convection-diffusion fluxes through each of the faces
!
$omp   parallel do private(IJK, IMJK, IJMK, IJKM, M)
    DO IJK = ijkstart3, ijkend3
      IF (FLUID_AT(IJK)) THEN
        IMJK = IM_OF(IJK)
        IJMK = JM_OF(IJK)
        IJKM = KM_OF(IJK)
!         B_M(IJK,0) = -((-((ROP_G(IJK)-ROP_GO(IJK))*VOL(IJK)*ODT+A_M(IJK,E,0&
!           )*U_G(IJK)-A_M(IJK,W,0)*U_G(IMJK)+A_M(IJK,N,0)*V_G(IJK)-A_M(IJK,&
!           S,0)*V_G(IJMK)+A_M(IJK,T,0)*W_G(IJK)-A_M(IJK,B,0)*W_G(IJKM)))+&
!           SUM_R_G(IJK)*VOL(IJK))
        bma = (ROP_G(IJK)-ROP_GO(IJK))*VOL(IJK)*ODT
                bme = A_M(IJK,E,0)*U_G(IJK)
        bmw = A_M(IJK,W,0)*U_G(IMJK)
        bmn = A_M(IJK,N,0)*V_G(IJK)
        bms = A_M(IJK,S,0)*V_G(IJMK)
        bmt = A_M(IJK,T,0)*W_G(IJK)
        bmb = A_M(IJK,B,0)*W_G(IJKM)
        bmr = SUM_R_G(IJK)*VOL(IJK)
        B_M(IJK,0) = -((-(bma + bme - bmw + bmn - bms + bmt - bmb ))+ bmr )
        B_MMAX(IJK,0) = max(abs(bma), abs(bme), abs(bmw), abs(bmn), abs(bms),
abs(bmt), abs(bmb), abs(bmr) )
        A_M(IJK,E,0) = A_M(IJK,E,0)*D_E(IJK,0)
        A_M(IJK,W,0) = A_M(IJK,W,0)*D_E(IMJK,0)
        A_M(IJK,N,0) = A_M(IJK,N,0)*D_N(IJK,0)
        A_M(IJK,S,0) = A_M(IJK,S,0)*D_N(IJMK,0)
        A_M(IJK,T,0) = A_M(IJK,T,0)*D_T(IJK,0)
        A_M(IJK,B,0) = A_M(IJK,B,0)*D_T(IJKM,0)
        DO M = 1, MMAX
          IF (.NOT.CLOSE_PACKED(M)) THEN
            B_M(IJK,0) = B_M(IJK,0) - ((-((ROP_S(IJK,M)-ROP_SO(IJK,M))*&
              VOL(IJK)*ODT+A_M(IJK,E,M)*U_S(IJK,M)-A_M(IJK,W,M)*U_S(IMJK&
              ,M)+A_M(IJK,N,M)*V_S(IJK,M)-A_M(IJK,S,M)*V_S(IJMK,M)+A_M(&
              IJK,T,M)*W_S(IJK,M)-A_M(IJK,B,M)*W_S(IJKM,M)))+SUM_R_S(IJK&
              ,M)*VOL(IJK))
            A_M(IJK,E,0) = A_M(IJK,E,0) + A_M(IJK,E,M)*D_E(IJK,M)
            A_M(IJK,W,0) = A_M(IJK,W,0) + A_M(IJK,W,M)*D_E(IMJK,M)
            A_M(IJK,N,0) = A_M(IJK,N,0) + A_M(IJK,N,M)*D_N(IJK,M)
            A_M(IJK,S,0) = A_M(IJK,S,0) + A_M(IJK,S,M)*D_N(IJMK,M)
            A_M(IJK,T,0) = A_M(IJK,T,0) + A_M(IJK,T,M)*D_T(IJK,M)
            A_M(IJK,B,0) = A_M(IJK,B,0) + A_M(IJK,B,M)*D_T(IJKM,M)
          ENDIF
        END DO
        A_M(IJK,0,0) = -(A_M(IJK,E,0)+A_M(IJK,W,0)+A_M(IJK,N,0)+A_M(IJK,S,0&
          )+A_M(IJK,T,0)+A_M(IJK,B,0))
!
        IF (ABS(A_M(IJK,0,0)) < SMALL_NUMBER) THEN
          IF (ABS(B_M(IJK,0)) < SMALL_NUMBER) THEN
            A_M(IJK,0,0) = -ONE
            B_M(IJK,0) = ZERO
          ELSE IF (RO_G0 .NE. UNDEFINED) THEN !This is an error only in incompressible
flow
$omp          critical
            WRITE (LINE, '(A,I6,A,I1,A,G12.5)') 'Error: At IJK = ', IJK, &
              ' M = ', 0, ' A = 0 and b = ', B_M(IJK,0)
            CALL WRITE_ERROR ('SOURCE_Pp_g', LINE, 1)
$omp          end critical
          ENDIF
        ENDIF
!
      ELSE
        A_M(IJK,E,0) = ZERO
        A_M(IJK,W,0) = ZERO
        A_M(IJK,N,0) = ZERO
        A_M(IJK,S,0) = ZERO
        A_M(IJK,T,0) = ZERO
        A_M(IJK,B,0) = ZERO
        A_M(IJK,0,0) = -ONE
        B_M(IJK,0) = ZERO
      ENDIF
    END DO

!loezos
    IF (SHEAR) THEN
$omp parallel do private(IJK)
              DO IJK = IJKSTART3, IJKEND3
      IF (FLUID_AT(IJK)) THEN
               V_G(IJK)=V_G(IJK)-VSH(IJK)
      END IF
     END DO
    END IF
!loezos

    IF (RO_G0 == UNDEFINED) THEN
     fac = UR_FAC(1)  !since p_g = p_g* + ur_fac * pp_g

!loezos
            incr=0
!loezos
!       CALL CALC_XSI(DISCRETIZE(1),ROP_G,U_G,V_G,W_G,XSI_E,XSI_N,XSI_T,incr)

$omp   parallel do                               &
$omp&   private(IJK,I,J,K,                        &
$omp&          IMJK,IJMK,IJKM,IJKE,IJKW,IJKN,IJKS,IJKT,IJKB)
      DO IJK = ijkstart3, ijkend3
        IF (FLUID_AT(IJK)) THEN

!
          A_M(IJK,0,0) = A_M(IJK,0,0) -
fac*DROODP_G(RO_G(IJK),P_G(IJK))*EP_G(IJK)*VOL(IJK)*ODT

!  Although the following is a better approximation for high speed flows because it
considers density changes
!  in the neighboring cells, the code runs faster without it for low speed flows.  The gas
phase mass balance
!  cannot be maintained to machine precision with the following approximation. If the
following lines are
!  uncommented, the calc_xsi call above should also be uncommented.
!
!           IMJK = IM_OF(IJK)
!           IJMK = JM_OF(IJK)
!           IJKM = KM_OF(IJK)
!           IJKE = EAST_OF(IJK)
!           IJKW = WEST_OF(IJK)
!           IJKN = NORTH_OF(IJK)
!           IJKS = SOUTH_OF(IJK)
!           IJKT = TOP_OF(IJK)
!           IJKB = BOTTOM_OF(IJK)
!           A_M(IJK,0,0) = A_M(IJK,0,0) - fac*DROODP_G(RO_G(IJK),P_G(IJK))*EP_G(&
!             IJK)*((ONE - XSI_E(IJK))*U_G(IJK)*AYZ(IJK)-XSI_E(IMJK)*U_G(&
!             IMJK)*AYZ(IMJK)+(ONE-XSI_N(IJK))*V_G(IJK)*AXZ(IJK)-XSI_N(IJMK&
!             )*V_G(IJMK)*AXZ(IJMK))
!
!           A_M(IJK,E,0) = A_M(IJK,E,0) - EP_G(IJKE)*fac*DROODP_G(RO_G(IJKE),P_G&
!             (IJKE))*XSI_E(IJK)*U_G(IJK)*AYZ(IJK)
!           A_M(IJK,W,0) = A_M(IJK,W,0) + EP_G(IJKW)*fac*DROODP_G(RO_G(IJKW),P_G&
!             (IJKW))*(ONE - XSI_E(IMJK))*U_G(IMJK)*AYZ(IMJK)
!           A_M(IJK,N,0) = A_M(IJK,N,0) - EP_G(IJKN)*fac*DROODP_G(RO_G(IJKN),P_G&
!             (IJKN))*XSI_N(IJK)*V_G(IJK)*AXZ(IJK)
!           A_M(IJK,S,0) = A_M(IJK,S,0) + EP_G(IJKS)*fac*DROODP_G(RO_G(IJKS),P_G&
!             (IJKS))*(ONE - XSI_N(IJMK))*V_G(IJMK)*AXZ(IJMK)
!           IF (DO_K) THEN
!             A_M(IJK,0,0) = A_M(IJK,0,0) - fac*DROODP_G(RO_G(IJK),P_G(IJK))*&
!               EP_G(IJK)*((ONE - XSI_T(IJK))*W_G(IJK)*AXY(IJK)-XSI_T(IJKM&
!               )*W_G(IJKM)*AXY(IJKM))
!             A_M(IJK,T,0) = A_M(IJK,T,0) - EP_G(IJKT)*fac*DROODP_G(RO_G(IJKT),&
!               P_G(IJKT))*XSI_T(IJK)*W_G(IJK)*AXY(IJK)
!             A_M(IJK,B,0) = A_M(IJK,B,0) + EP_G(IJKB)*fac*DROODP_G(RO_G(IJKB),&
!               P_G(IJKB))*(ONE - XSI_T(IJKM))*W_G(IJKM)*AXY(IJKM)
!           ENDIF
!
          ENDIF
        END DO
      ENDIF

!   Remove the asymmetry in matrix caused by the pressure outlet or inlet boundaries.
Because
!   the P' at such boundaries is zero we may set the coefficient in the neighboring
!   fluid cell to zero without affecting the linear equation set.
$omp   parallel do                               &
$omp&   private(IJK,IMJK, IPJK, IJMK, IJPK, IJKM, IJKP)
      DO IJK = ijkstart3, ijkend3
        IF (FLUID_AT(IJK)) THEN
          IMJK = IM_OF(IJK)
          IPJK = IP_OF(IJK)
          IJMK = JM_OF(IJK)
          IJPK = JP_OF(IJK)
          IJKM = KM_OF(IJK)
          IJKP = KP_OF(IJK)
                if(p_flow_at(imjk))A_m(IJK, w, 0) = ZERO
                if(p_flow_at(ipjk))A_m(IJK, e, 0) = ZERO
                if(p_flow_at(ijmk))A_m(IJK, s, 0) = ZERO
                if(p_flow_at(ijpk))A_m(IJK, n, 0) = ZERO
                if(p_flow_at(ijkm))A_m(IJK, b, 0) = ZERO
                if(p_flow_at(ijkp))A_m(IJK, t, 0) = ZERO

        ENDIF
      END DO
!
!  Specify P' to zero at a certain location for incompressible flows and
!  cyclic boundary conditions.
!
!// Parallel implementation of fixing a pressure at a point
   I = I_OF_G(IJK_P_G)
   J = J_OF_G(IJK_P_G)
   K = K_OF_G(IJK_P_G)

   IF(IS_ON_myPE_OWNS(I,J,K)) THEN
     IF (IJK_P_G /= UNDEFINED_I) THEN
       IJK_P_G_LOCAL = FUNIJK(I,J,K)
         A_M(IJK_P_G_LOCAL,E,0) = ZERO
         A_M(IJK_P_G_LOCAL,W,0) = ZERO
         A_M(IJK_P_G_LOCAL,N,0) = ZERO
         A_M(IJK_P_G_LOCAL,S,0) = ZERO
         A_M(IJK_P_G_LOCAL,T,0) = ZERO
         A_M(IJK_P_G_LOCAL,B,0) = ZERO
         A_M(IJK_P_G_LOCAL,0,0) = -ONE
         B_M(IJK_P_G_LOCAL,0) = ZERO
       ENDIF
     ENDIF
!
!     CHEM & ISAT begin (nan xie)
      IF (CALL_DI .or. CALL_ISAT) THEN
        SUM_R_G = SUM_R_G_temp
        SUM_R_S = SUM_R_S_temp
      END IF
!     CHEM & ISAT end (nan xie)
      call unlock_xsi_array

      RETURN
      END SUBROUTINE SOURCE_PP_G

!// Comments on the modifications for DMP version implementation
!// 001 Include header file and common declarations for parallelization
!// 350 Changed do loop limits: 1,ijkmax2-> ijkstart3, ijkend3
```

- MPI-callable, OpenMP-enabled.
- 340 Fortran lines.
- No MPI-specific code.
- Ubiquitous OpenMP markup (red regions).

MFIX: Multiphase Flows with Interphase eXchanges (https://www.mfix.org/)

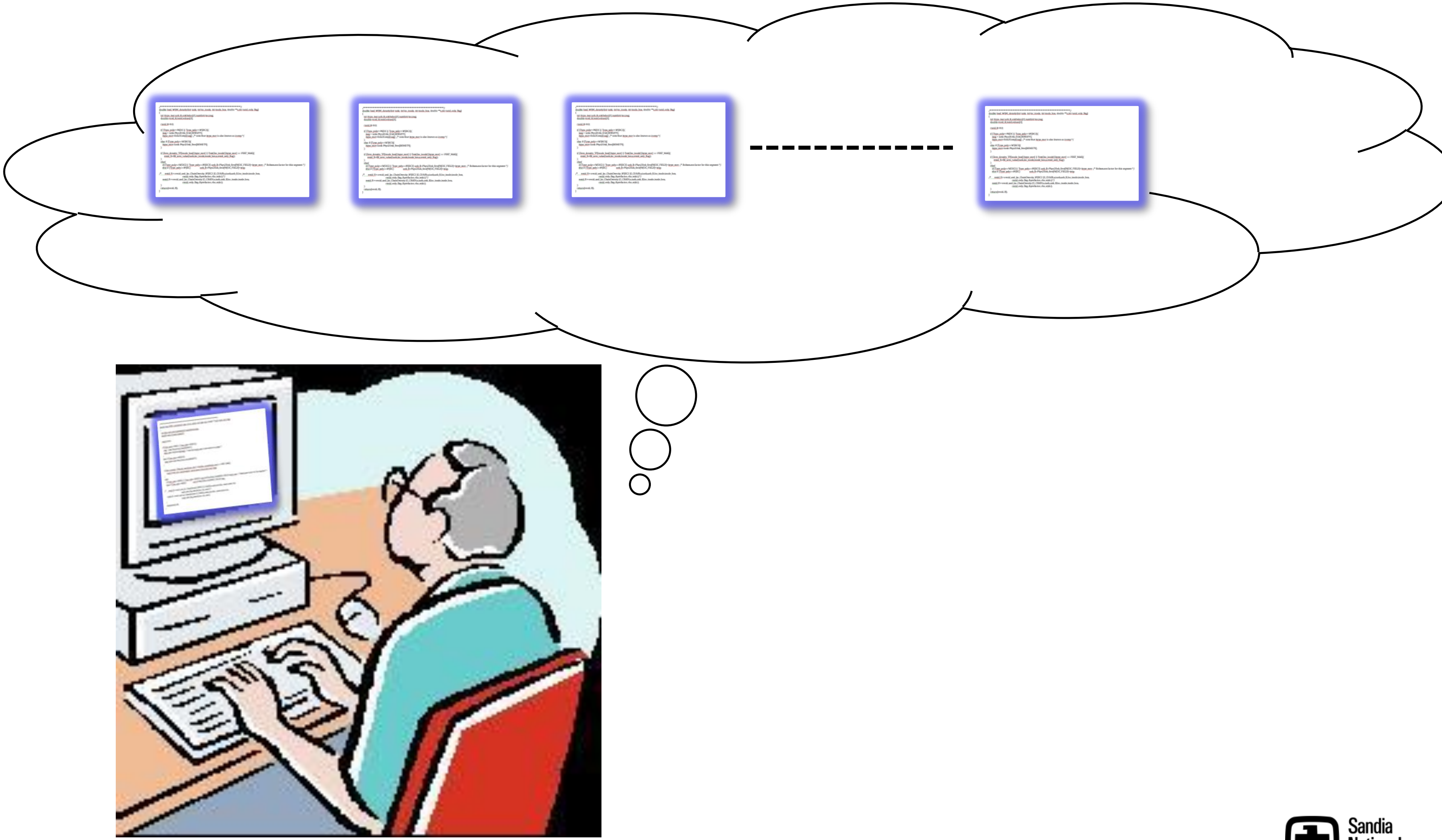
Reasons for MPI Success?

- Portability? Yes.
- Standardized? Yes.
- Momentum? Yes.
- Separation of many Parallel & Algorithms concerns? Big Yes.

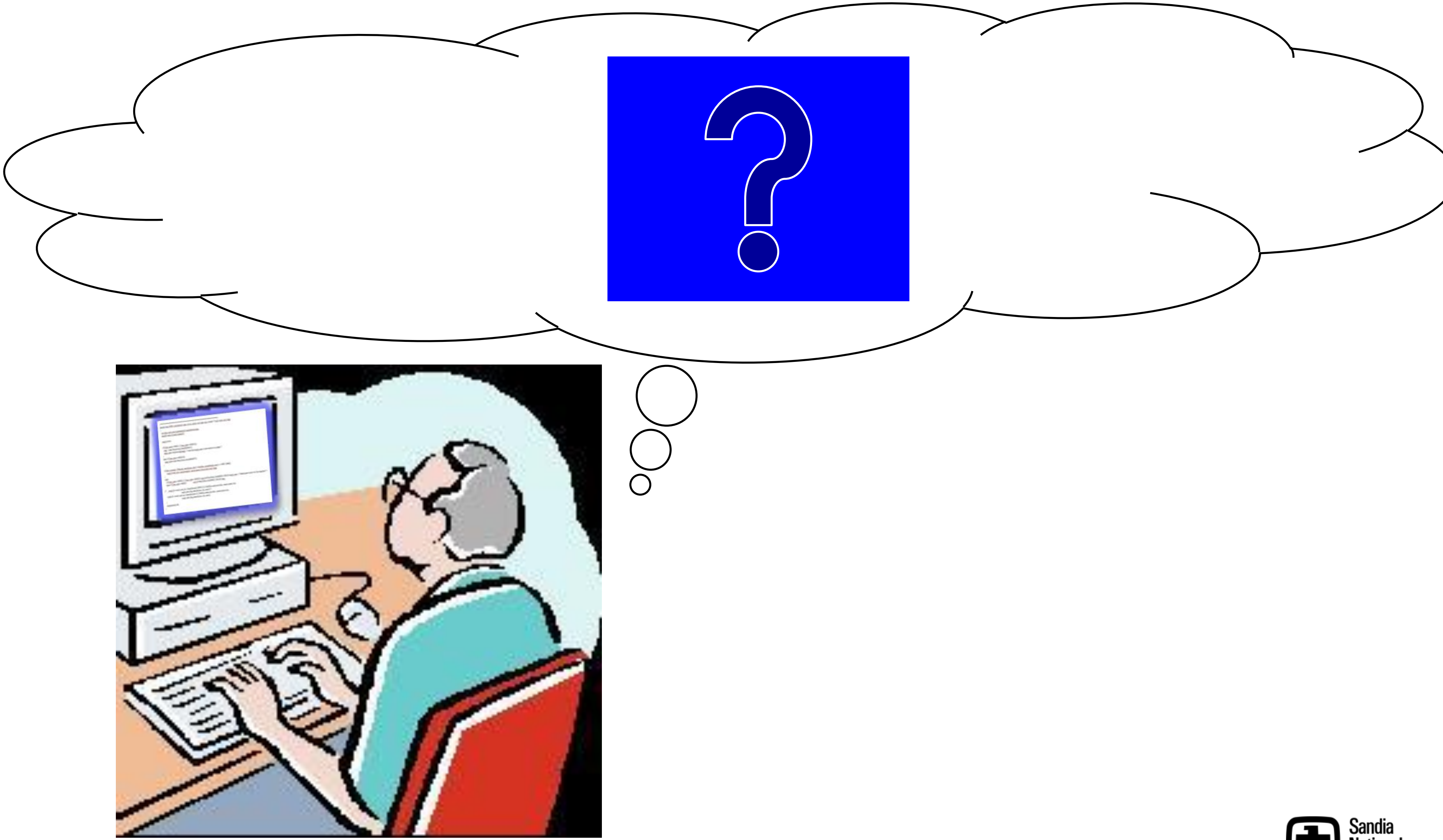
- Once framework in place:
  - Sophisticated physics added as serial code.
  - Ratio of science experts vs. parallel experts: 10:1.

- Key goal for new parallel apps: Preserve this ratio

Computational Domain Expert Writing MPI Code

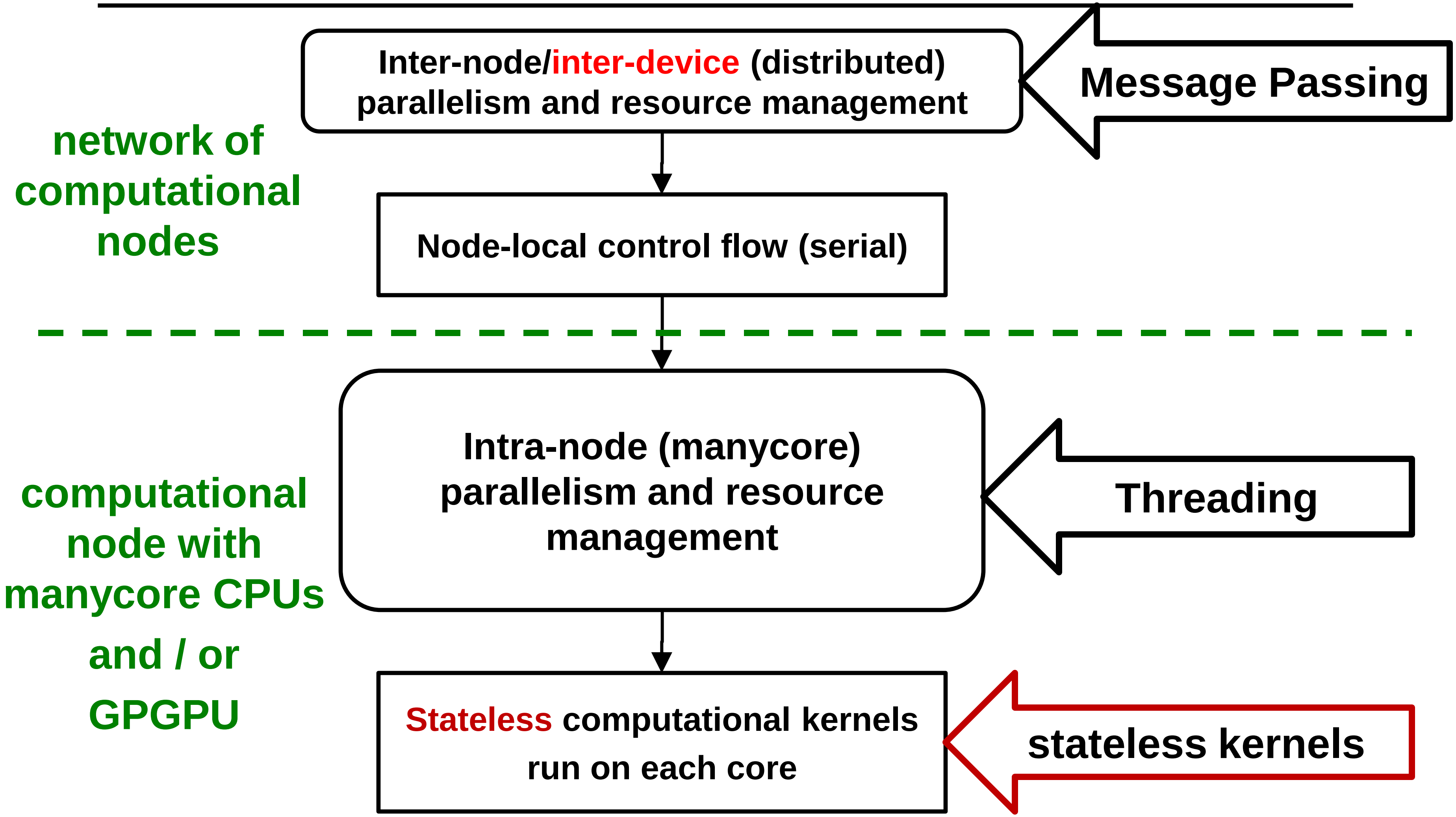


Computational Domain Expert Writing Future Parallel Code



Evolving Parallel Programming Model

Parallel Programming Model: Multi-level/Multi-device



network of computational nodes

Inter-node/inter-device (distributed) parallelism and resource management

Message Passing

Node-local control flow (serial)

computational node with manycore CPUs

and / or

GPGPU

Intra-node (manycore) parallelism and resource management

Threading

Stateless computational kernels run on each core

stateless kernels

Adapted from slide of H. Carter Edwards

Domain Scientist’s Parallel Palette

- MPI-only (SPMD) apps:
  - Single parallel construct.
  - Simultaneous execution.
  - Parallelism of even the messiest serial code.
- Next-generation applications:
  - Internode:
    - MPI, yes, or something like it.
    - Composed with intranode.
  - Intranode:
    - Much richer palette.
    - More care required from programmer.
- What are the constructs in our new palette?

Obvious Constructs/Concerns

- Parallel for:
  - No loop-carried dependence.
  - Rich loops.
- Parallel reduce:
  - Couple with other computations.
  - Concern for reproducibility.

Other construct: Pipeline

- Sequence of filters.
- Each filter is:
  - Sequential (grab element ID, enter global assembly) or
  - Parallel (fill element stiffness matrix).
- Filters executed in sequence.
- Programmer’s concern:
  - Determine (conceptually): Can filter execute in parallel?
  - Write filter (serial code).
  - Register it with the pipeline.
- Extensible:
  - New physics feature.
  - New filter added to pipeline.

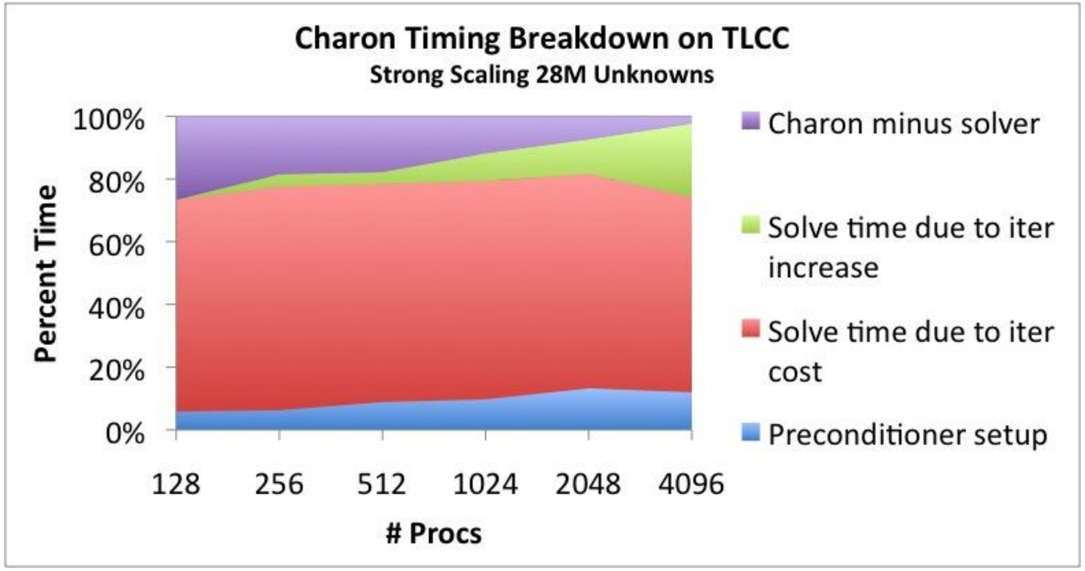
Other construct: Thread team

- Multiple threads.
- Fast barrier.
- Shared, fast access memory pool.
- Example: Nvidia SM
- X86 more vague, emerging more clearly in future.

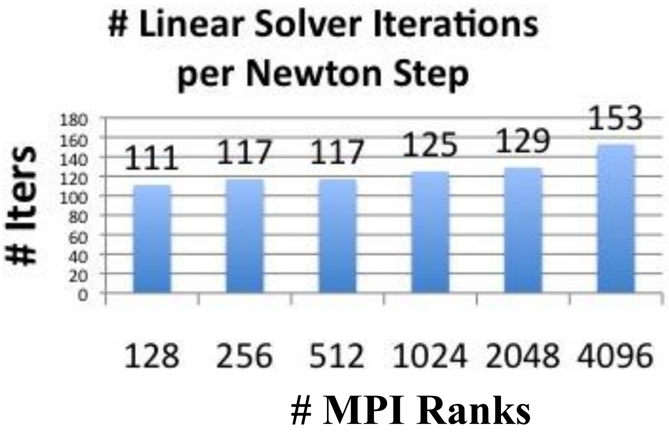
Finite Elements/Volumes/Differences and parallel node constructs

- Parallel for, reduce, pipeline:
  - Sufficient for vast majority of node level computation.
  - Supports:
    - Complex modeling expression.
    - Vanilla parallelism.
- Thread team:
  - Complicated.
  - Requires true parallel algorithm knowledge.
  - Useful in solvers.

Preconditioners for Scalable Multicore Systems



Strong scaling of Charon on TLCC (P. Lin, J. Shadid 2009)

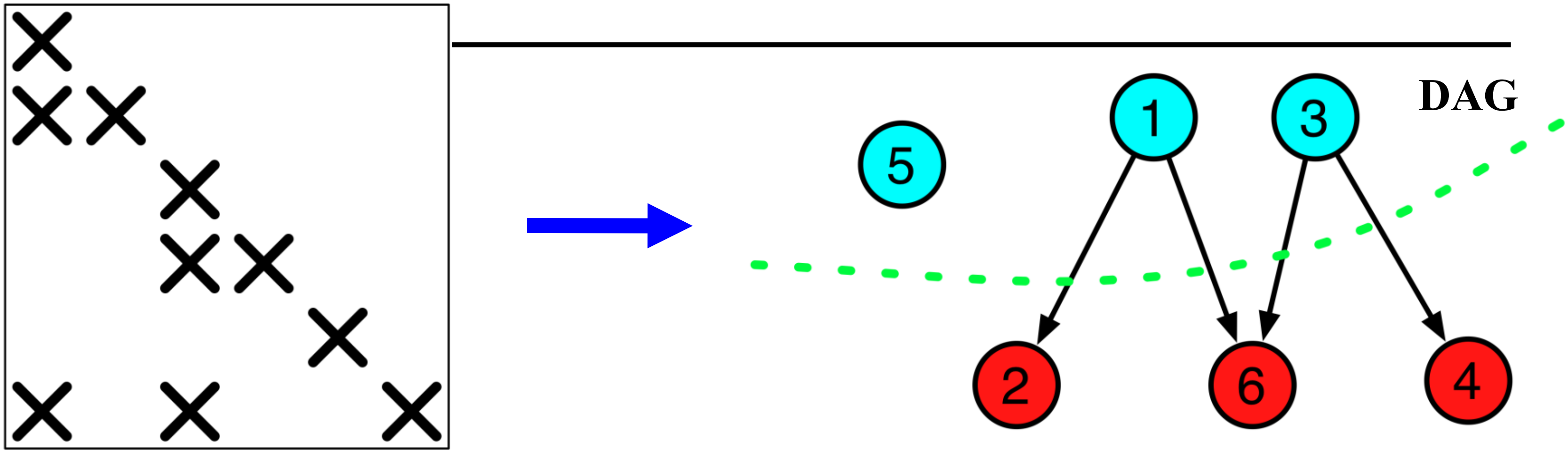


- Observe: Iteration count increases with number of subdomains.
- With scalable threaded triangular solves
  - Solve triangular system on larger subdomains.
  - Reduce number of subdomains.
- Goal:
  - Better kernel scaling (threads vs. MPI processes).
  - Better convergence, More robust.
- Note: App (-solver) scales very well in MPI-only mode.
- Exascale Potential: Tiled, pipelined implementation.

MPI Tasks	Threads	Iterations
4096	1	153
2048	2	129
1024	4	125
512	8	117
256	16	117
128	32	111

***Factors Impacting Performance of Multithreaded Sparse Triangular Solve,* Michael M. Wolf and Michael A. Heroux and Erik G. Boman, VECPAR 2010, to appear.**

Level Set Triangular Solver



L

Triangular Solve:

- Critical Kernel
  - MG Smoothers
  - Incomplete IC/ILU
- Naturally Sequential
- Building on classic algorithms:
  - Level Sched:
    - circa 1990.
    - Vectorization.
  - New: Generalized.

$$\tilde{L} = PLP^T = \begin{bmatrix} D_1 & & & & \\ A_{2,1} & D_2 & & & \\ A_{3,1} & A_{3,2} & D_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ A_{l,1} & A_{l,2} & A_{l,3} & \ldots & D_l \end{bmatrix}$$

Permuted System

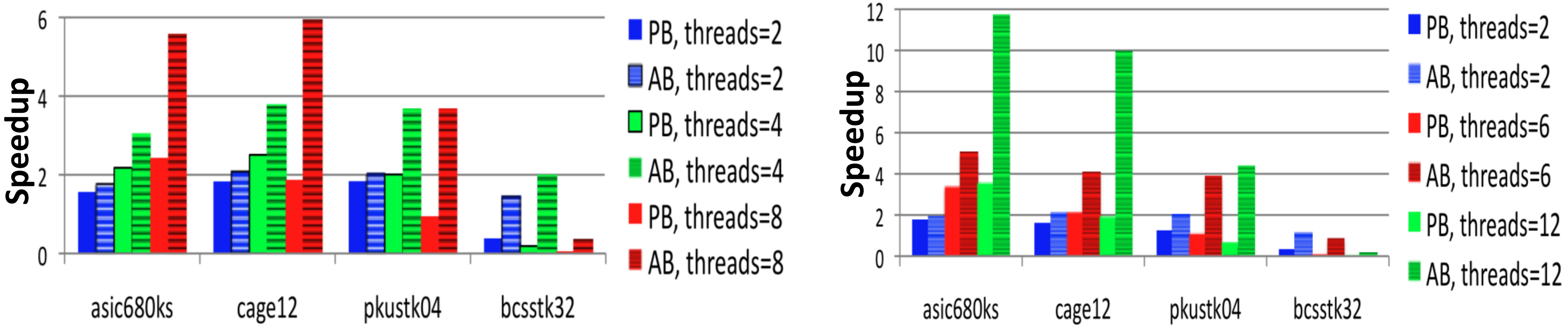
$$\begin{aligned} \tilde{x}_1 &= D_1^{-1}\tilde{y}_1 \\ \tilde{x}_2 &= D_2^{-1}(\tilde{y}_2 - A_{2,1}\tilde{x}_1) \\ \vdots \quad & \vdots \qquad \vdots \\ \tilde{x}_l &= D_l^{-1}(\tilde{y}_l - A_{l,1}\tilde{x}_1 - \ldots - A_{l,l-1}\tilde{x}_{l-1}) \end{aligned}$$

Multi-step Algorithm

Triangular Solve Results

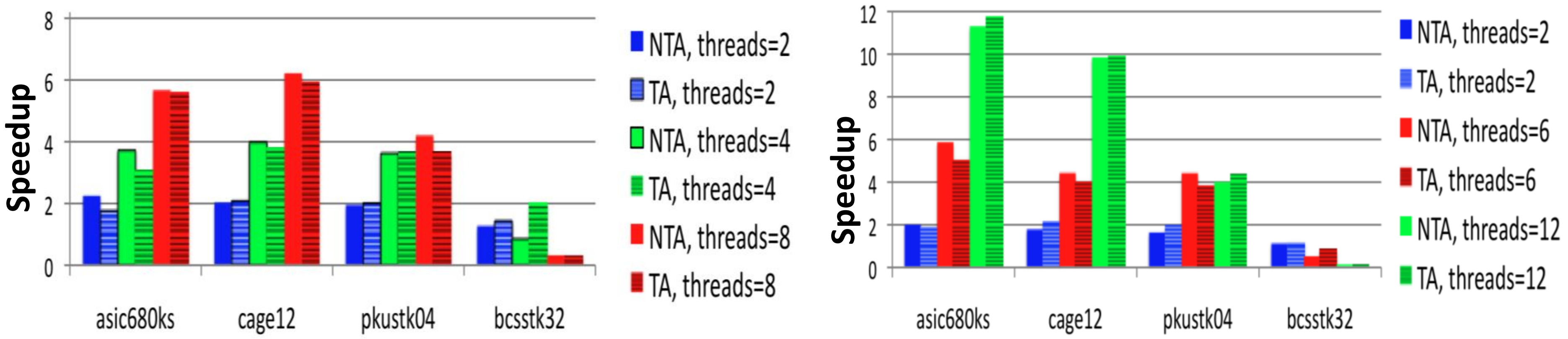
Name	N	nnz	N/nlevels	application area
asic680ks	682,712	2,329,176	13932.9	circuit simulation
cage12	130,228	2,032,536	1973.2	DNA electrophoresis
pkustk04	55,590	4,218,660	149.4	structural engineering
bcsstk32	44,609	2,014,701	15.1	structural engineering

Passive (PB) vs. Active (AB) Barriers: Critical for Performance



Nehalem

Istanbul



AB + No Thread Affinity (NTA) vs. AB + Thread Affinity (TA) : Also Helpful

Level sets: Trilinos/Isorropia Core Kernel Timings: Trilinos/Kokkos.

Thread Team Advantanges

- **Qualitatively better algorithm:**
  - **Threaded triangular solve scales.**
  - **Fewer MPI ranks means fewer iterations, better robustness.**
- **Exploits:**
  - **Shared data.**
  - **Fast barrier.**
  - **Data-driven parallelism.**

Placement and Migration

Placement and Migration

- MPI:
  - Data/work placement clear.
  - Migration explicit.
- Threading:
  - It’s a mess (IMHO).
  - Some platforms good.
  - Many not.
  - Default is bad (but getting better).
  - Some issues are intrinsic.

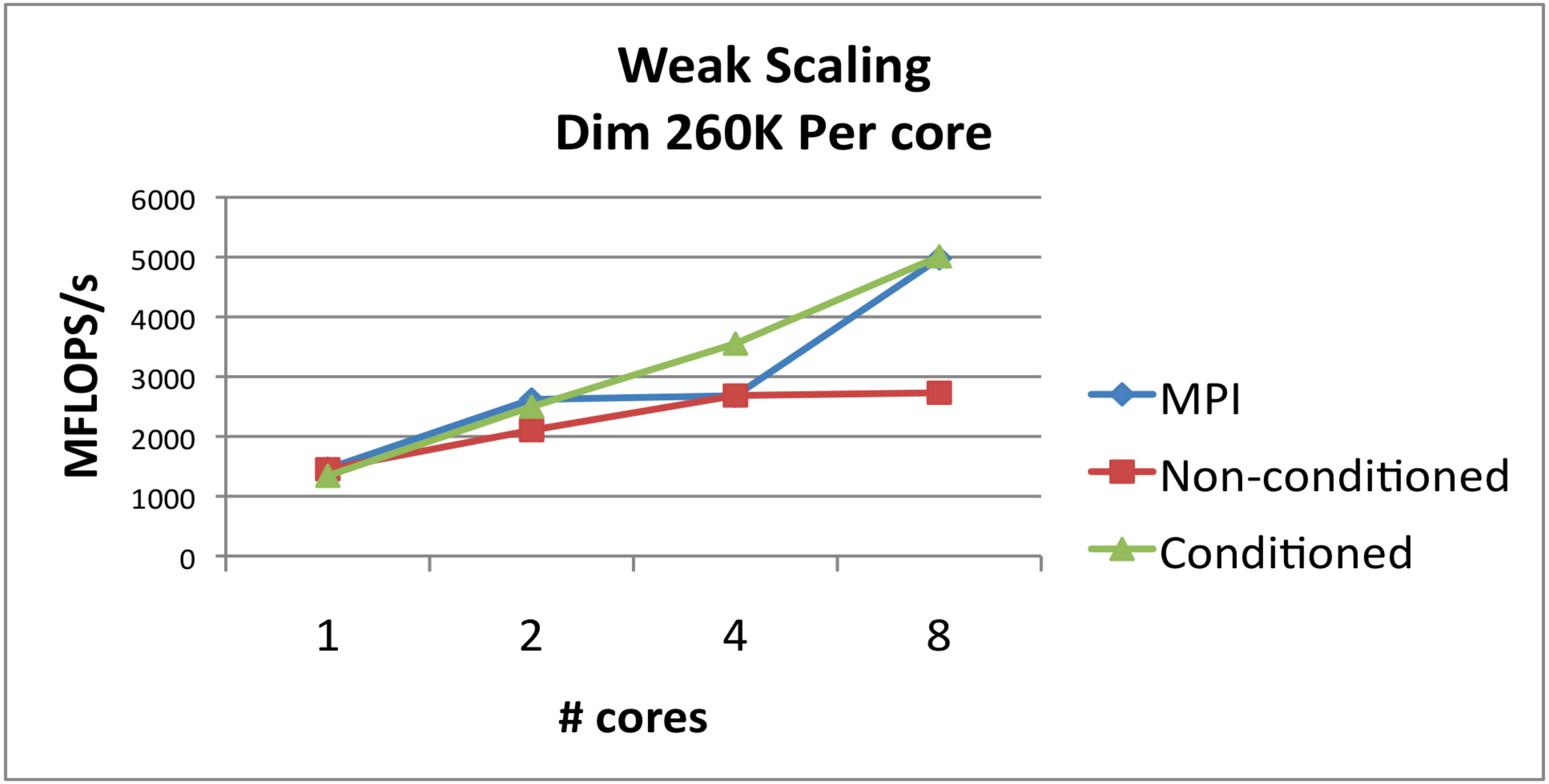
Data Placement on NUMA

- Memory Intensive computations: Page placement has huge impact.
- Most systems: First touch (except LWKs).
- Application data objects:
  - Phase 1: Construction phase, e.g., finite element assembly.
  - Phase 2: Use phase, e.g., linear solve.
- Problem: First touch difficult to control in phase 1.
- Idea: Page migration.
  - Not new: SGI Origin. Many old papers on topic.

Data placement experiments

- MiniApp: HPCCG (Mantevo Project)
- Construct sparse linear system, solve with CG.
- Two modes:
  - Data placed by assembly, not migrated for NUMA
  - Data migrated using parallel access pattern of CG.
- Results on dual socket quad-core Nehalem system.

Weak Scaling Problem



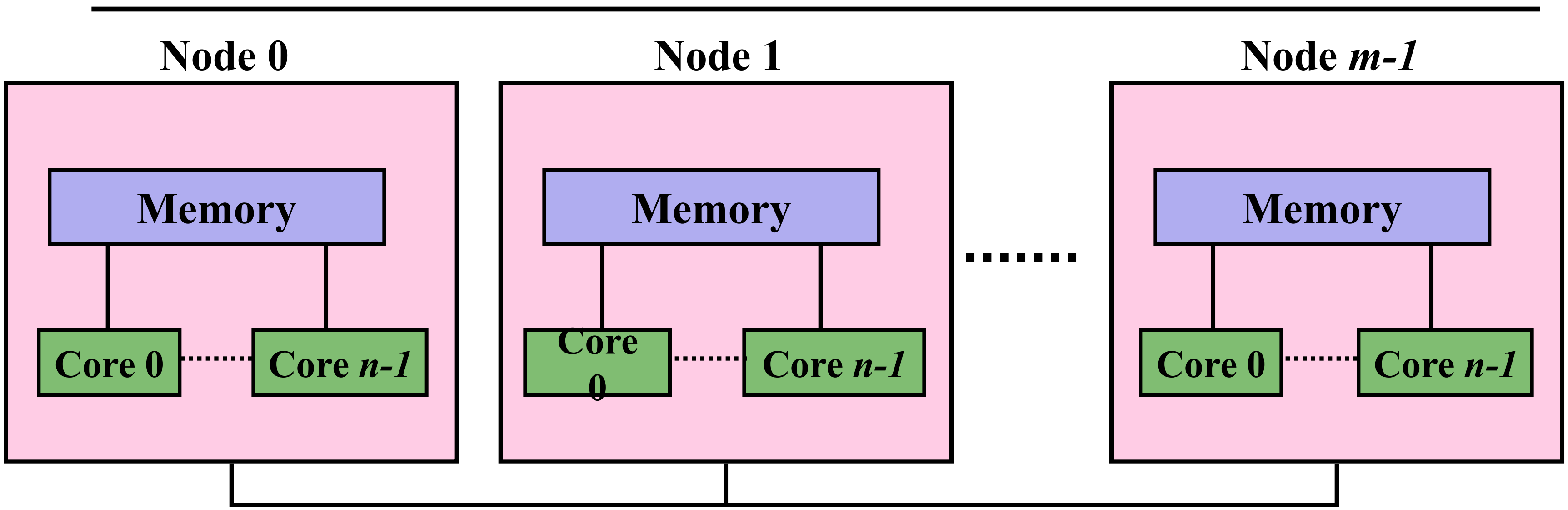
- MPI and conditioned data approach comparable.
- Non-conditioned very poor scaling.

Page Placement summary

- MPI+OpenMP (or any threading approach) is best overall.
- But:
  - Data placement is big issue.
  - Hard to control.
  - Insufficient runtime support.
- Current work:
  - Migrate on next-touch (MONT).
  - Considered in OpenMP (next version).
  - Also being studied in Kitten (Kevin Pedretti).
- Note: This phenomenon especially damaging to OpenMP common usage.

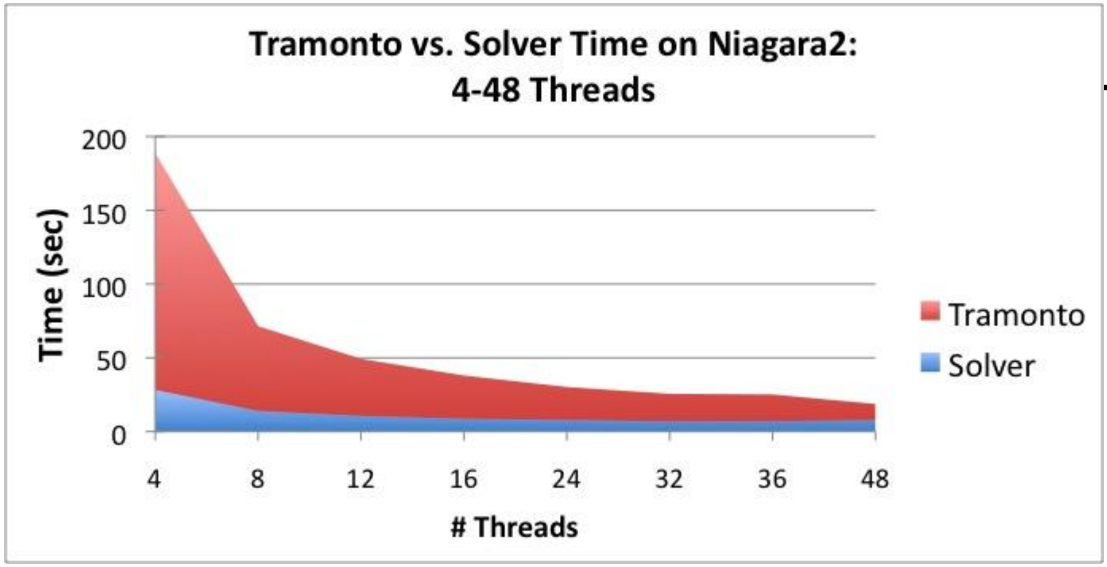
Transition: MPI-only to MPI+[X|Y|Z]

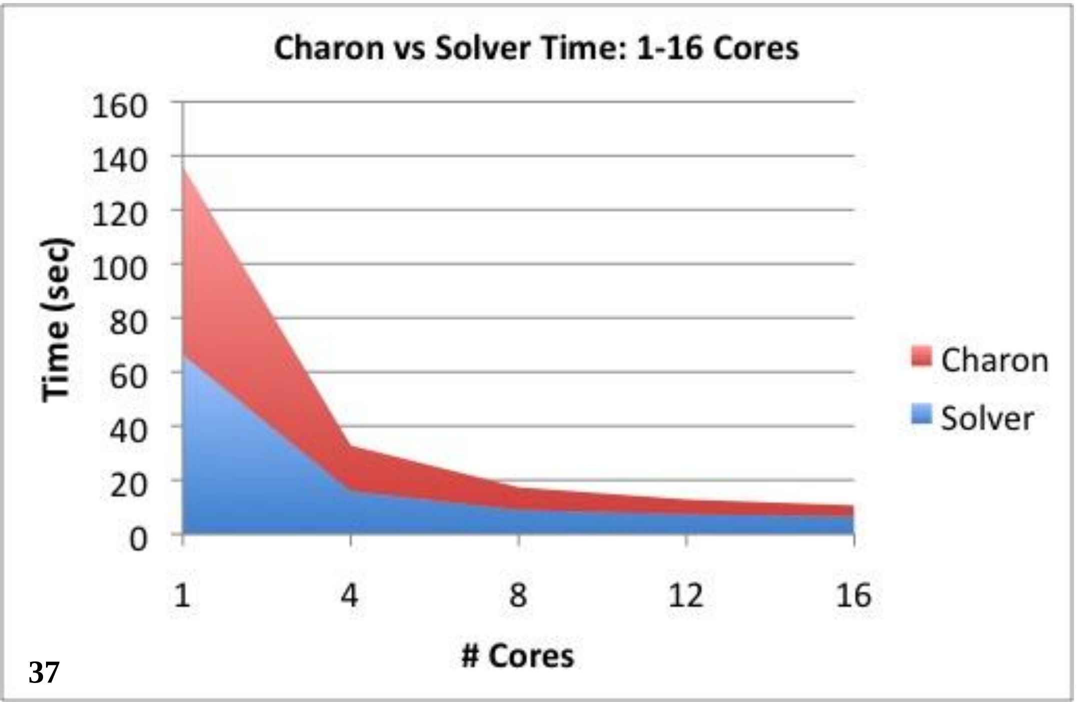
Parallel Machine Block Diagram



– Parallel machine with $p = m * n$ processors:
  - m = number of nodes.
  - n = number of shared memory processors per node.
– Two ways to program:
  - Way 1: p MPI processes.
  - Way 2: m MPI processes with n threads per MPI process.
- New third way:
  - “Way 1” in some parts of the execution (the app).
  - “Way 2” in others (the solver).

Multicore Scaling: App vs. Solver





Application:

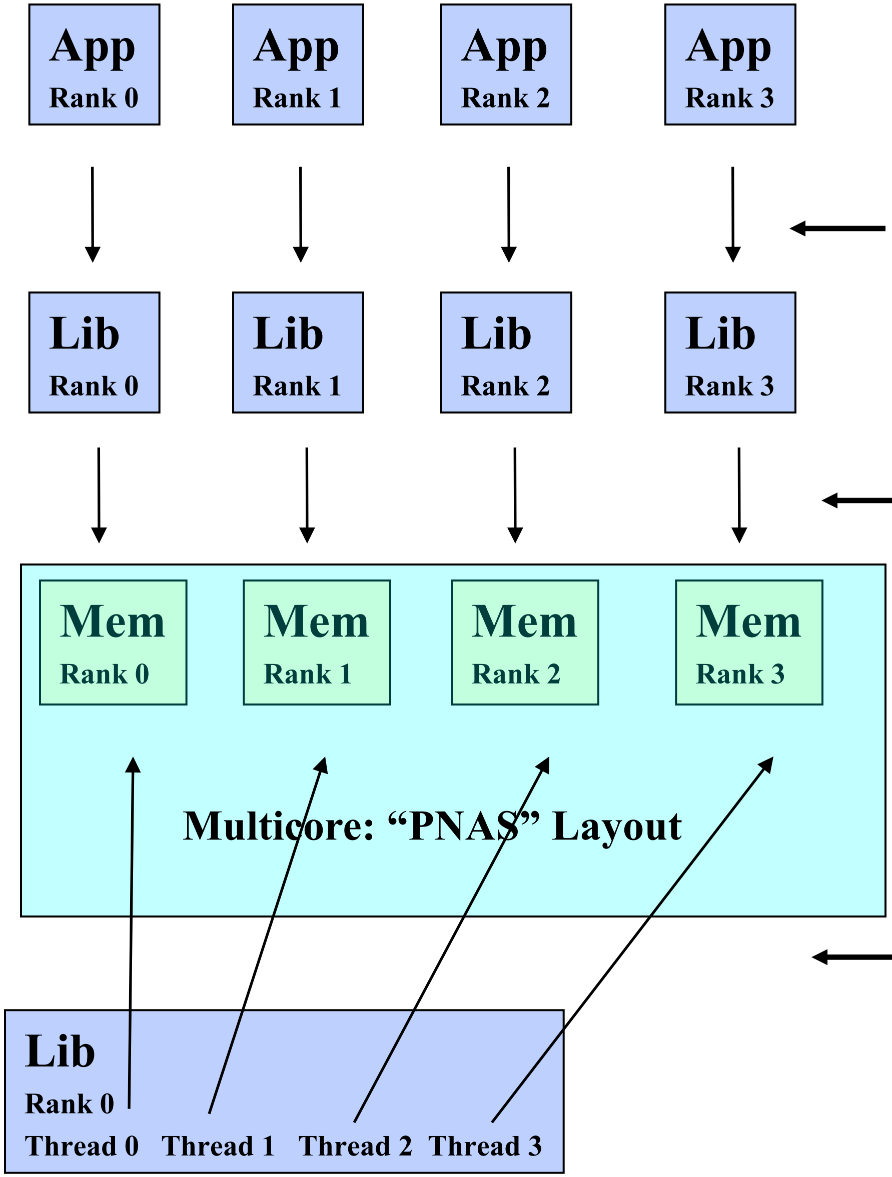
- Scales well (sometimes superlinear)
- MPI-only sufficient.

Solver:

- Scales more poorly.
- Memory system-limited.
- MPI+threads can help.

* Charon Results:
Lin & Shadid TLCC Report

MPI-Only + MPI/Threading: *Ax=b*



App passes matrix and vector values to library data classes

All ranks store *A, x, b* data in memory visible to rank 0

Library solves *Ax=b* using shared memory algorithms on the node.

MPI Shared Memory Allocation

Idea:

- Shared memory alloc/free functions:
  - MPI_Comm_alloc_mem
  - MPI_Comm_free_mem
- Predefined communicators:

  MPI_COMM_NODE – ranks on node

  MPI_COMM_SOCKET – UMA ranks

  MPI_COMM_NETWORK – inter node
- Status:
  - Available in current development branch of OpenMPI.
  - First “Hello World” Program works.
  - Incorporation into standard still not certain. Need to build case.
  - Next Step: Demonstrate usage with threaded triangular solve.
- Exascale potential:
  - Incremental path to MPI+X.
  - Dial-able SMP scope.

```
int n = …;
double* values;
 MPI_Comm_alloc_mem(
                MPI_COMM_NODE,  // comm (SOCKET works too)
                n*sizeof(double),        // size in bytes
                MPI_INFO_NULL,      // placeholder for now
                &values);                  // Pointer to shared array (out)

// At this point:
// - All ranks on a node/socket have pointer to a shared buffer (values).
// - Can continue in MPI mode (using shared memory algorithms) or
// - Can quiet all but one:
int rank;
MPI_Comm_rank(MPI_COMM_NODE, &rank);
if (rank==0) { // Start threaded code segment, only on rank 0 of the node
…
}

 MPI_Comm_free_mem(MPI_COMM_NODE, values);
```

Collaborators: B. Barrett, Brightwell, Wolf - SNL; Vallee, Koenig - ORNL

Resilient Algorithms

My Luxury in Life (wrt FT/Resilience)

The privilege to think of a computer as a *reliable, digital* machine.

“At 8 nm process technology, it will be harder to tell a 1 from a 0.”

(W. Camp 2008, 2010)

Users’ View of the System Now

- “All nodes up and running.”
- Certainly nodes fail, but invisible to user.
- No need for me to be concerned.
- Someone else’s problem.

Users’ View of the System
Future

- Nodes in one of four states.
  1. Dead.
  2. Dying (perhaps producing faulty results).
  3. Reviving.
  4. Running properly:
     a) Fully reliable or…
     b) Maybe still producing an occasional bad result.

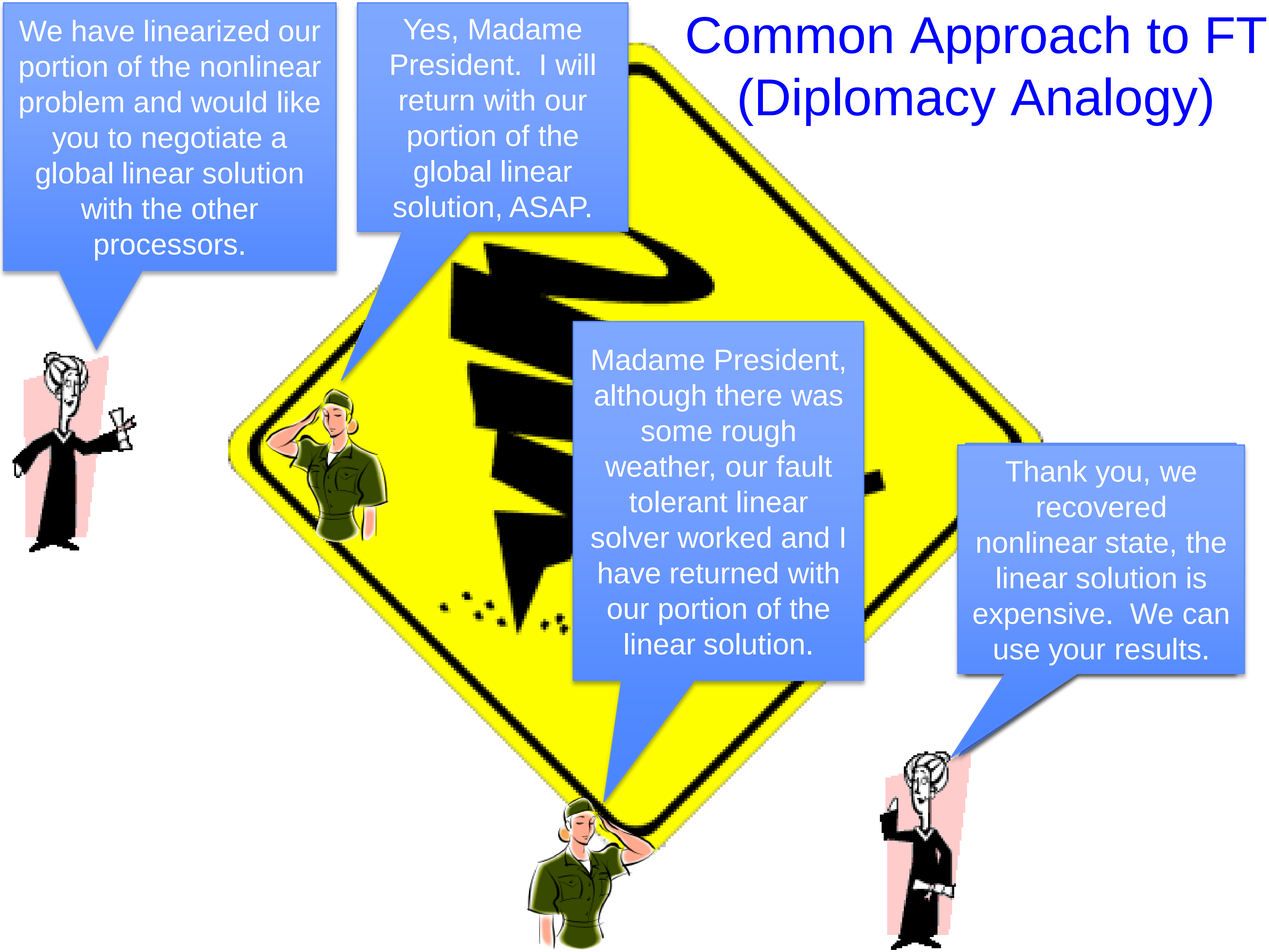
Faults: Hard vs. Soft

- Hard:
  - Program flow interrupted.
  - Majority of faults.
  - Presently handled by (global) checkpoint/restart.
  - Numerous papers on alternatives.
- Soft:
  - Program flow continues.
  - Minor perturbations in data state:
    - Incorrect address lookup (but still in user scope).
    - Incorrect FP value.

Algorithm-Based (Hard) Fault Tolerance

- Numerous approaches.
- Most common strategies:
  - Meta data:
    - Embed meta data into user-defined data structures.
    - Manage fault detection, recovery manually.
  - Algorithm results validation:
    - Use known algorithm properties.
    - Validate computed to known (e.g., residual check).
- Note: A lack of app awareness.

Common Approach to FT (Diplomacy Analogy)



We have linearized our portion of the nonlinear problem and would like you to negotiate a global linear solution with the other processors.

Yes, Madame President. I will return with our portion of the global linear solution, ASAP.

Madame President, although there was some rough weather, our fault tolerant linear solver worked and I have returned with our portion of the linear solution.

Thank you, we recovered nonlinear state, the linear solution is expensive. We can use your results.

Hard Error Futures

- C/R will continue as dominant approach:
  - – Global state to global file system OK for small systems.
  - – Large systems: State control will be localized, use SSD.
- Checkpoint-less restart:
  - – Requires full vertical HW/SW stack co-operation.
  - – Very challenging.
  - – Stratified research efforts not effective.

Soft Error Futures

- Soft error handling: A legitimate algorithms issue.
- Programming model, runtime environment play role.

Consider GMRES as an example of how soft errors affect correctness

- Basic Steps
  1) Compute Krylov subspace (preconditioned sparse matrix-vector multiplies)
  2) Compute orthonormal basis for Krylov subspace (matrix factorization)
  3) Compute vector yielding minimum residual in subspace (linear least squares)
  4) Map to next iterate in the full space
  5) Repeat until residual is sufficiently small

- More examples in Bronevetsky & Supinski, 2008

Why GMRES?

- Many apps are implicit.
- Most popular (nonsymmetric) linear solver is preconditioned GMRES.
- Only small subset of calculations need to be reliable.
  - GMRES is iterative, but also direct.

Every calculation matters

- Small PDE Problem: Dim 21K, Nz 923K.
- ILUT/GMRES
- Correct computation 35 Iters: 343M FLOPS
- Two examples of a single bad floating point op

Description	Iterations	FLOPS	Recursive Residual Error	Solution Error
All Correct Calcs	35	343M	4.6e-15	1.0e-6
Iter=2, y[1] += 1.0 SpMV incorrect Ortho subspace	35	343M	6.7e-15	3.7e+3
Q[1][1] += 1.0 Non-ortho subspace	N/C	N/A	7.7e-02	5.9e+5

One possible approach is transactional computation

- Database transactions: atomic
- Transactional memory: atomic memory operation
- Transactional computation:
  - Designated sensitive computation region (orthogonalization step in GMRES)
  - Guarantee accurate computation or notify user.

Needs to be coupled with SW-enabled guaranteed data regions

- User-designated reliable data region
- Extra protection to improve reliable data storage and transfer
- Examples
  - Original input data (needed for verification)
  - Linear solver: *A*, *x*, *b*
  - Orthogonal vectors for GMRES
- OpenMP pragma-enabled?

Goal

- Algorithms well-conditioned wrt soft failure.
- Now:
  - Single soft error produces erroneous results.
- Goal:
  - Correct results always.
  - Cost increase proportional to number of soft errors.
- Note: These are just two approaches to ABFT.

Software Development and Delivery

“Are C++ templates safe? No, but they are good.”

Compile-time Polymorphism

Templates and Sanity upon a shifting foundation

Software delivery:

- Essential Activity

How can we:

- Implement mixed precision algorithms?
- Implement generic fine-grain parallelism?
- Support hybrid CPU/GPU computations?
- Support extended precision?
- Explore redundant computations?
- Prepare for both exascale “swim lanes”?

C++ templates only sane way:

- Moving to completely templated Trilinos libraries.
- Other important benefits.
- **A usable stack exists now in Trilinos.**

Template Benefits:

- – Compile time polymorphism.
- – True generic programming.
- – No runtime performance hit.
- – Strong typing for mixed precision.
- – Support for extended precision.
- – Many more…

Template Drawbacks:

- – Huge compile-time performance hit:
  - But good use of multicore :)
  - Eliminated for common data types.
- - Complex notation:
  - - Esp. for Fortran & C programmers).
  - - Can insulate to some extent.

Solver Software Stack

🟪 Phase I packages: SPMD, int/double

🟩 Phase II packages: Templated

Optimization **Unconstrained:** **Constrained:**	Find $u \in \Re^n$ that minimizes $g(u)$ Find $x \in \Re^m$ and $u \in \Re^n$ that minimizes $g(x,u)$ s.t. $f(x,u) = 0$	**Sensitivities** **(Automatic Differentiation: Sacado)**	🟩 MOOCHO
Bifurcation Analysis	Given nonlinear operator $F(x,u) \in \Re^{n+m}$ For $F(x,u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$		🟪 LOCA
Transient Problems **DAEs/ODEs:**	Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0,T], x(0) = x_0, \dot{x}(0) = x_0'$ for $x(t) \in \Re^n, t \in [0,T]$		🟩 Rythmos
Nonlinear Problems	Given nonlinear operator $F(x) \in \Re^m \to \Re$ Solve $F(x) = 0$ $x \in \Re^n$		🟪 NOX
Linear Problems **Linear Equations:** **Eigen Problems:**	Given Linear Ops (Matrices) $A, B \in \Re^{m \times n}$ Solve $Ax = b$ for $x \in \Re^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \Re^n$, $\lambda \in$		🟩 Anasazi 🟪 Ifpack, ML, etc... 🟪 AztecOO
Distributed Linear Algebra **Matrix/Graph Equations:** **Vector Problems:**	Compute $y = Ax; A = A(G); A \in \Re^{m \times n}, G \in \Im^{m \times n}$ Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \Re^n$		🟪 Epetra 🟩 Teuchos

Solver Software Stack

Phase I packages	Phase II packages	Phase III packages: Manycore*, templated

Optimization Unconstrained: Constrained:	Find $u \in \Re^n$ that minimizes $g(u)$ Find $x \in \Re^m$ and $u \in \Re^n$ that minimizes $g(x,u)$ s.t. $f(x,u) = 0$	Sensitivities (Automatic Differentiation: Sacado)	MOOCHO	
Bifurcation Analysis	Given nonlinear operator $F(x,u) \in \Re^{n+m}$ - For $F(x,u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$		LOCA	T-LOCA
Transient Problems DAEs/ODEs:	Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0,T], x(0) = x_0, \dot{x}(0) = x_0^{'}$ for $x(t) \in \Re^n, t \in [0,T]$		Rythmos	
Nonlinear Problems	Given nonlinear operator $F(x) \in \Re^m \to \Re$ Solve $F(x) = 0$ $x \in \Re^n$		NOX	T-NOX
Linear Problems Linear Equations: Eigen Problems:	Given Linear Ops (Matrices) $A, B \in \Re^{m \times n}$ Solve $Ax = b$ for $x \in \Re^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \Re^n$, $\lambda \in$		Anasazi AztecOO Ifpack, ML, etc...	Anasazi Belos* T-Ifpack*, T-ML*, etc.
Distributed Linear Algebra Matrix/Graph Equations: Vector Problems:	Compute $y = Ax; A = A(G); A \in \Re^{m \times n}, G \in \Im^{m \times n}$ Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \Re^n$		Epetra	Tpetra* Kokkos
			Teuchos	

Trilinos/Kokkos Node API

Generic Shared Memory Node

- Abstract inter-node comm provides DMP support.
- Need some way to portably handle SMP support.
- Goal: allow code, once written, to be run on any parallel node, regardless of architecture.
- Difficulty #1: Many different memory architectures
  - Node may have multiple, disjoint memory spaces.
  - Optimal performance may require special memory placement.
- Difficulty #2: Kernels must be tailored to architecture
  - Implementation of optimal kernel will vary between archs
  - No universal binary ➔ need for separate compilation paths

Sandia National Laboratories

Kokkos Node API

- Kokkos provides two main components:
  - Kokkos memory model addresses Difficulty #1
    - Allocation, deallocation and efficient access of memory
    - compute buffer: special memory used for parallel computation
    - New: Local Store Pointer and Buffer with size.
  - Kokkos compute model addresses Difficulty #2
    - Description of kernels for parallel execution on a node
    - Provides stubs for common parallel work constructs
    - Currently, parallel for loop and parallel reduce
- Code is developed around a polymorphic Node object.
- Supporting a new platform requires only the implementation of a new node type.

Kokkos Memory Model

- A generic node model must at least:
  - support the scenario involving distinct device memory
  - allow efficient memory access under traditional scenarios
- Nodes provide the following memory routines:

```
ArrayRCP<T> Node::allocBuffer<T>(size_t sz);
void        Node::copyToBuffer<T>(  T * src,
                                    ArrayRCP<T>  dest);
void        Node::copyFromBuffer<T>(ArrayRCP<T> src,
                                    T * dest);
ArrayRCP<T> Node::viewBuffer<T> (ArrayRCP<T> buff);
void        Node::readyBuffer<T>(ArrayRCP<T> buff);
```

Kokkos Compute Model

- How to make shared-memory programming generic:
  - Parallel reduction is the intersection of `dot()` and `norm1()`
  - Parallel for loop is the intersection of `axpy()` and mat-vec
  - We need a way of fusing kernels with these basic constructs.
- Template meta-programming is the answer.
  - This is the same approach that Intel TBB and Thrust take.
  - Has the effect of requiring that Tpetra objects be templated on Node type.
- Node provides generic parallel constructs, user fills in the rest:

```
template <class WDP>
void Node::parallel_for(
  int beg, int end, WDP workdata);
```

Work-data pair (WDP) struct provides:

- loop body via `WDP::execute(i)`

```
template <class WDP>
WDP::ReductionType Node::parallel_reduce(
  int beg, int end, WDP workdata);
```

Work-data pair (WDP) struct provides:

- reduction type `WDP::ReductionType`
- element generation via `WDP::generate(i)`
- reduction via `WDP::reduce(x,y)`

Example Kernels: axpy() and dot()

```
template <class WDP>
void
Node::parallel_for(int beg, int end,
                   WDP workdata    );
```

```
template <class T>
struct AxpyOp {
  const T * x;
  T * y;
  T alpha, beta;
  void execute(int i)
  { y[i] = alpha*x[i] + beta*y[i]; }
};
```

```
AxpyOp<double> op;
op.x = ...;  op.alpha = ...;
op.y = ...;  op.beta  = ...;
node.parallel_for< AxpyOp<double> >
                 (0, length, op);
```

```
template <class WDP>
WDP::ReductionType
Node::parallel_reduce(int beg, int end,
                      WDP workdata    );
```

```
template <class T>
struct DotOp {
  typedef T ReductionType;
  const T * x, * y;
  T identity()       { return (T)0;       }
  T generate(int i)  { return x[i]*y[i]; }
  T reduce(T x, T y) { return x + y;      }
};
```

```
DotOp<float> op;
op.x = ...;  op.y = ...;
float dot;
dot = node.parallel_reduce< DotOp<float> >
                          (0, length, op);
```

Hybrid CPU/GPU Computing

Hybrid Timings (Tpetra)

- Tests of a simple iterations:
  - power method: one sparse mat-vec, two vector operations
  - conjugate gradient: one sparse mat-vec, five vector operations
- DNVS/x104 from UF Sparse Matrix Collection (100K rows, 9M entries)
- NCCS/ORNL Lens node includes:
  - one NVIDIA Tesla C1060
  - one NVIDIA 8800 GTX
  - Four AMD quad-core CPUs
- Results are very tentative!
  - suboptimal GPU traffic
  - bad format/kernel for GPU
  - bad data placement for threads

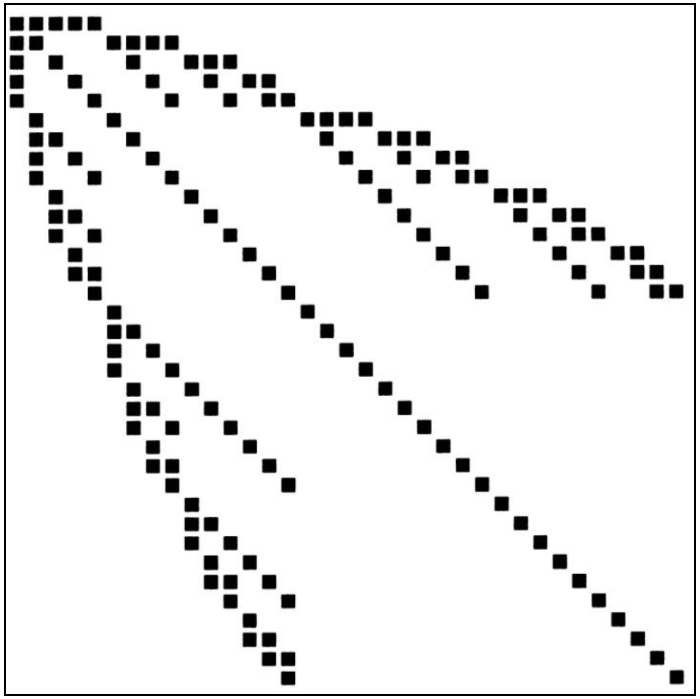
Node	PM (mflop/s)	CG (mflop/s)
Single thread	140	614
8800 GPU	1,172	1,222
Tesla GPU	1,475	1,531
Tesla + 8800	981	1,025
16 threads	816	1,376
1 node 15 threads + Tesla	867	1,731
2 nodes 15 threads + Tesla	1,677	2,102

New Core Linear Algebra Needs

Advanced Modeling and Simulation Capabilities: Stability, Uncertainty and Optimization

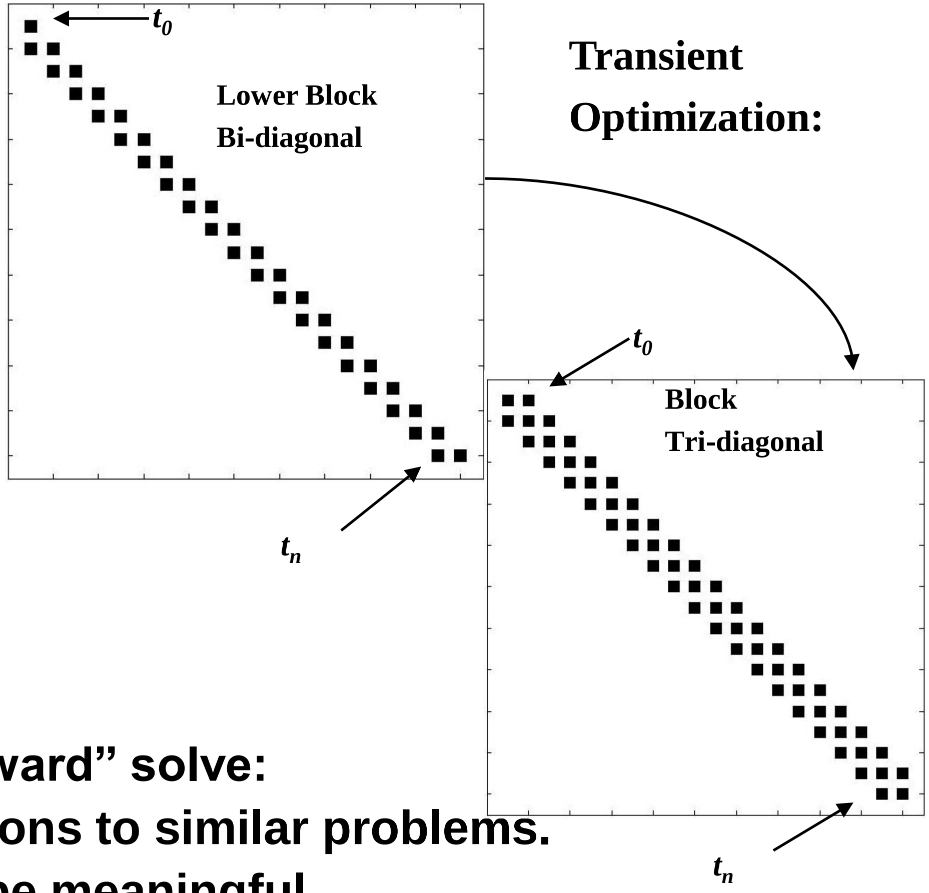
- **Promise: 10-1000 times increase in parallelism (or more).**

SPDEs:



■ →

Transient Optimization:



- **Pre-requisite: High-fidelity "forward" solve:**
  - **Computing families of solutions to similar problems.**
  - **Differences in results must be meaningful.**

■ - Size of a single forward problem

Advanced Capabilities: Readiness and Importance

Modeling Area	**Sufficient Fidelity?**	**Other concerns**	**Advanced capabilities priority**
Seismic *S. Collis, C. Ober*	Yes.	None as big.	Top.
Shock & Multiphysics (Alegra) *A. Robinson, C. Ober*	Yes, but some concerns.	Constitutive models, material responses maturity.	Secondary now. Non-intrusive most attractive.
Multiphysics (Charon) *J. Shadid*	Reacting flow w/ simple transport, device w/ drift diffusion, …	Higher fidelity, more accurate multiphysics.	Emerging, not top.
Solid mechanics *K. Pierson*	Yes, but…	Better contact. Better timestepping. Failure modeling.	Not high for now.

Advanced Capabilities:
Other issues

- Non-intrusive algorithms (e.g., Dakota):
  - Task level parallel:
    - A true peta/exa scale problem?
    - Needs a cluster of 1000 tera/peta scale nodes.
- Embedded/intrusive algorithms (e.g., Trilinos):
  - Cost of code refactoring:
    - Non-linear application becomes “subroutine”.
    - Disruptive, pervasive design changes.
- Forward problem fidelity:
  - Not uniformly available.
  - Smoothness issues.
  - Material responses.

Advanced Capabilities: Derived Requirements

- Large-scale problem presents collections of related subproblems with forward problem sizes.
- Linear Solvers: $Ax = b \rightarrow AX = B, \quad Ax^i = b^i, \quad A^i x^i = b^i$
  - Krylov methods for multiple RHS, related systems.
- Preconditioners: $A^i = A_0 + \Delta A^i$
  - Preconditioners for related systems.
- Data structures/communication: $pattern(A^i) = pattern(A^j)$
  - Substantial graph data reuse.

Summary

- App targets will change:
  - Advanced modeling and simulation: Gives a better answer.
  - Kernel set changes.
- Resilience requires an integrated strategy:
  - Most effort at the system/runtime level.
  - C/R (with localization) will continue at the app level.
  - Resilient algorithms will mitigate soft error impact.
- Building the next generation of parallel applications requires enabling domain scientists:
  - Write sophisticated methods.
  - Do so with serial fragments.
  - Fragments hoisted into scalable, resilient fragment.

Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have MPI_Init().
3. All future programmers will need to write parallel code.
4. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
5. DRY is not possible across CPUs and GPUs
6. GPUs are a harbinger of CPU things to come.
7. Checkpoint/Restart will be sufficient for scalable resilience.
8. Resilience will be built into algorithms.
9. MPI-only and MPI+X can coexist in the same application.
10. Kernels will be different in the future.